Kurzanleitung PLCcom.Opc.Ua.Sdk Version 6 für .Net

Indi.An Systems GmbH

Flughafenallee 3
D-28199 Bremen
support@indi-systems.de

Tel + 49 421-989703-30
Fax + 49 421-989703-39

Inhaltsverzeichnis

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Über dieses Dokument?

Das vorliegende Dokument soll Ihnen einen ersten Überblick über die bereitgestellten Funktionalitäten liefern. Es handelt sich nicht um eine komplette Dokumentation, sondern wird bereitgestellt um Ihnen einen ersten Einstieg zu ermöglichen. Weitergehende Informationen entnehmen Sie bitte

- den Code-Beispielen im Auslieferungspaket,
- den Code-Beispielen und der Onlinehilfe auf unserer Website http://www.plccom.net/help_opcua_toolkit/help/index.html
- sowie der jeweiligen Onlinebeschreibung im Auslieferungspaket (index.html)

Wichtiger Sicherheitshinweis

Mit der Software PLCcom.Opc.Ua.Sdk werden Sie oder der Anwender in die Lage versetzt nach eigenem Ermessen Anlagen, Maschinen oder Ähnliches zu überwachen und zu steuern. Hierzu muss der Benutzer eigenes Wissen bzw. diverse Tätigkeiten einfließen lassen.

Bevor das hieraus resultierende Arbeitsergebnis an der Anlage, Maschine oder Ähnlichem eingesetzt werden kann, muss der Ersteller eines Projektes sämtliche Funktionen getestet und auf einwandfreie Funktion und einwandfreies Zusammenspiel mit der Anlage, Maschine oder Ähnlichem überprüft haben. Diese Tests sind nach jeder Software-Änderung sowie nach jeder Änderung an der Anlage, Maschine oder Ähnlichem bzw. der Peripherie (Netzwerk, Server, etc.) zu wiederholen.

Sollten Fehlfunktionen auftreten oder erkannt werden, darf das PLCcom.Opc.Ua.Sdk nicht an der Anlage, Maschine oder Ähnlichem betrieben werden.

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen Tel + 49 421-989703-30
support@indi-systems.de Fax + 49 421-989703-39

Was ist eigentlich OPC.UA?

Mit der OPC UA Spezifikation stellt die OPC-Foundation ein neuentwickeltes Kommunikationsmodell zum einheitlichen Transport von Maschinendaten zur Verfügung. Ziel war es, dass OPC-kommunikationsmodell an die Erfordernisse zukünftiger Anwendungen anzupassen, und die bestehenden Nachteile der auf DCOM basierenden OPC-Schnittstelle auszugleichen. Das OPC UA Kommunikationsmodell ist eine komplette Neuentwicklung, unterscheidet sich erheblich von seinem Vorgänger und basiert auch nicht auf der DCOM-Schnittstelle.
Die erste Version der OPC UA Spezifikation wurde im Jahre 2006 zur Verfügung gestellt, eine Überarbeitung fand im Jahr 2009 statt.

Mit OPC UA wird ein zukunftsträchtiger einheitlicher Kommunikationsstandard bereitgestellt, der auch die Erfordernisse von Industrie 4.0 Anwendungen abdeckt.

Was ist das PLCcom.Opc.Ua.Sdk?

Das PLCcom.Opc.Ua.Sdk ist eine speziell für .NET-Softwareentwickler bereitgestellte hoch optimierte und moderne Komponente, um Softwareentwicklern komfortabel Zugriff auf eine clientseitige OPC-UA-Schnittstelle zur Verfügung zu stellen, z.B. um Daten auszulesen oder zu schreiben.

Bei der Software PLCcom.Opc.Ua.Sdk handelt es sich um den direkten Nachfolger bzw. Weiterentwicklung des PLCcom.Opc.Ua.Toolkit.

Die Library wird für das .net-Framework und als .net Standard-Komponente zur Verfügung gestellt und steht den Entwicklern auch für plattformübergreifende .net Core oder Xamarin-Anwendungen zur Verfügung.

Bei den Librarys handelt es sich je nach Version um 100%.Net-Dateien. Die Komponente kann direkt als Verweis eingebunden werden, API-Aufrufe sind nicht notwendig. Es ist problemlos möglich, die Komponenten in 32-oder 64 Bit-Umgebungen sowie plattformübergreifend einzusetzen. Die internen Routinen sind auf High-Performance-Zugriffe optimiert.

Mit dem PLCcom.Opc.Ua.Sdk können sie Applikationen erstellen, die die gängigsten OPC-Spezifikationen unterstützen.
Unter anderem sind das:

- DataAccess (am meisten verwendet)
- Alarm & Conditions
- Historical Data
- Historical Events

Mit im Lieferumfang enthalten sind umfangreiche Code-Beispiele und Tutorials enthalten, die die leichte Anbindung eines OPC-UA-Servers über eine OPC-Schnittstelle an Ihre Applikation verdeutlichen und auch in Ihren Projekten genutzt werden können.

Für die Entwicklungsunterstützung sind Testserver- und client-Applikationen im Auslieferungspaket enthalten.

Welche Vorteile bringt mir das PLCcom.Opc.Ua.Sdk?

Mit dem Sdk werden .Net-Entwickler in die Lage versetzt, mit wenigen Zeilen Code ihre entwickelten Anwendungen einen standardisierten OPC.UA-Clientzugriff hinzuzufügen und hiermit auf bestehende OPC-UA-Server-Instanzen zuzugreifen.

Bei der Entwicklung des Toolkits haben den Focus auf eine schnelle Erlernbarkeit und Einsetzbarkeit gelegt. Aus diesem Grunde wurden für die meisten Funktionalitäten einfache vorkonfigurierte Befehle zur Verfügung gestellt.

Die Adressierung der OPC-Nodes kann im Toolkit auch über den Browse-Namen als String durchgeführt werden, das aufwendige Ermitteln der NodeIDs führt das PLCcom.Opc.Ua.Sdk für Sie im Hintergrund aus z.B. "Objects.Server.Data.Static.Scalar.Int64Value"

Nachfolgend einige Beispiele zur Veranschaulichung der einfachen Implementierung von OPC UA Funktionen.

Beispiel Abfragen von bestehenden Endpoints eines Servers:

```
EndpointDescriptionCollection Endpoints =
         UaClient.GetEndpoints(new Uri("opc.tcp://localhost:4048"));
```

Beispiel Lesen einer Variable:

```
var value = client.ReadValue("Objects.Server.Data.Static.Scalar.Int64Value");
```

Beispiel Monitoring einer Variable:

```
{
//create a new subscription
Subscription subscription = new Subscription();

//add new subscription to client
client.AddSubscription(subscription);

//Create a monitoring item and add to the subscription
string browsePath = "Objects.Server.Data.Dynamic.Scalar.Int64Value";
NodeId nodeId = client.GetNodeIdByPath(browsePath);
MonitoredItem monitoredItem = new MonitoredItem(subscription.DefaultItem)
          {
           StartNodeId = nodeId,
           DisplayName = nodeId.ToString()
          };

//register monitoring event
monitoredItem.Notification += Client_MonitorNotification;
//add Item to subscription
subscription.AddItem(monitoredItem);

//apply changes
subscription.ApplyChanges();
}
```

```
private void Client_MonitorNotification(MonitoredItem monitoredItem, MonitoredItemNotification
EventArgs e)
{
```

```
 MonitoredItemNotification notification = e.NotificationValue as MonitoredItemNotification;
 Console.WriteLine(monitoredItem.StartNodeId.Identifier + " Value: " + notification.Value +
 " Status: " + notification.Value.StatusCode.ToString());
}
```

Weitere Vorteile sind:

- Einfache Handhabung, viele Funktionen lassen sich mit einer einzigen Zeile Code abbilden
- Per Default **automatische Connect-, Reconnect- sowie Disconnect-Funktionalitäten**, der Verbindungszustand braucht vom Entwickler nicht überwacht werden
- Der Serverzustand wird per aktiver **Keepalive-Überwachung** verfolgt
- **Umfangreiche Tutorials** für einen schnellen Einstieg in den .Net-Sprachen C# und Visual Basic im Auslieferungspaket
- Für die schnelle Einarbeitung und Tests werden Beispiel OPC-Server und OPC-Clients installiert

Welche Voraussetzungen sind für die Erstellung von Anwendungen mit der PLCcom.Opc.Ua.Sdk notwendig?

Zur Erstellung von Anwendungen mit dem Toolkit sind fortgeschrittene Programmierkenntnisse in einer .Net-Programmiersprache vorteilhafterweise mit C# oder Visual Basic notwendig.

Für den Betrieb der PLCcom.Opc.Ua.Sdk sind außerdem folgende System-Komponenten Voraussetzung:

- Microsoft .net Framework 4.7.2 oder neuer
- Microsoft .net Core Framework 3.0 oder neuer
- Microsoft .net 5.0 oder neuer
- Eine Entwicklungsumgebung welche mit .net Standard 2.1 kompatibel ist (z.B. Xamarin)

Um die beigefügten Programmbeispiele ausführen zu können, benötigen Sie folgende Programmierwerkzeuge:

- Visual Studio 2017 oder neuer
- Ein Code-Editor einer Entwicklungsumgebung, die kompatibel mit .net Standard 2.1 ist (z.B. Xamarin)

Wie übergebe ich die Lizensierungsinformationen?

Das PLCcom.Opc.Ua.Sdk muss durch Eingabe von Lizenzinformationen freigeschaltet werden. Diese Lizenzinformationen wurden Ihnen entweder nach dem Kauf oder durch Zusendung eines 30-Tage-Demo-Keys übermittelt.

Die Übergabe der Lizenzinformationen erfolgt während der Erstellung einer Client-Instanz.

```
UaClient client = new UaClient(<Enter your UserName here>, <Enter your Serial here>);
```

Wie benutze ich die Discovery-Funktionalitäten?

Die Kommunikation zwischen UA-Client und UA-Server wird über sogenannte Endpunkte durchgeführt, die der jeweilige OPC-UA-Server zur Verfügung stellt.

Zur Herstellung einer clientseitigen Verbindung müssen entweder die Endpunkt-Informationen des OPC-UA-Servers bekannt sein, oder diese Informationen können mit den Discovery-Funktionalitäten der PLCcom.Opc.Ua.Sdk ermittelt werden.
Hierzu muss die URL des Opc-Servers bekannt sein.

```
//C#
string url = "opc.tcp://localhost:50520/UA/DataAccessServer";

//get all servers
ApplicationDescriptionCollection servers = UaClient.FindServers(new Uri(url));
```

```
'Visual Basic'
Dim url As String = "opc.tcp://localhost:50520/UA/DataAccessServer"

'get all servers'
Dim servers As ApplicationDescriptionCollection = UaClient.FindServers(New Uri(url))
```

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Wie erstelle ich eine neue Client Instanz?

Zum Erstellen einer neuen Client-Instanz muss zuerst eine Session-Konfiguration erstellt und parametrisiert werden. Für eine einfache Sessionkonfiguration ist die Übergabe des Server-Endpoints und der Timeout-Zeitspanne ausreichend.

```
//C#
//get the endpoints
EndpointDescriptionCollection ep = UaClient.GetEndpoints(new Uri("opc.tcp://localhost:4048"),
60000);

//create a a SessionConfiguration with the selected endpoint and application name
SessionConfiguration sessionConfiguration = SessionConfiguration.Build("Appname", ep[0]);
```

```
'Visual Basic'
'get the endpoints'
Dim ep As EndpointDescriptionCollection = UaClient.GetEndpoints(New Uri("opc.tcp://localhost:4
048"), 60000)

'create a a SessionConfiguration with the selected endpoint and application name'
Dim sessionConfiguration As SessionConfiguration = SessionConfiguration.Build("Appname",ep(0))
```

Im zweiten Schritt wird die Client-Instanz erstellt und das Sessionkonfiguration-Objekt übergeben:

```
//C#
UaClient client = new UaClient("<Enter your UserName here>",
                               "<Enter your Serial here>",
                               sessionConfiguration);
```

```
'Visual Basic'
Dim client As UaClient = New UaClient("<Enter your UserName here>",
                                      "<Enter your Serial here>",
                                      sessionConfiguration)
```

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Zum Überwachen der Session können Events registriert werden:

```
//C#
client.ServerConnectionLost += Client_ServerConnectionLost;
client.ServerConnected += Client_ServerConnected;
client.KeepAlive += Client_KeepAlive;

private void Client_ServerConnected(object sender, EventArgs e)
{
   Console.WriteLine(DateTime.Now.ToLocalTime() + " Session connected");
}

private void Client_ServerConnectionLost(object sender, EventArgs e)
{
   Console.WriteLine(DateTime.Now.ToLocalTime() + " Session connection lost");
}

void Client_KeepAlive(Session session, KeepAliveEventArgs e)
{
   //catch the keepalive event
}
```

```
'Visual Basic'
AddHandler client.ServerConnectionLost, AddressOf Client_ServerConnectionLost
AddHandler client.ServerConnected, AddressOf Client_ServerConnected
AddHandler client.KeepAlive, AddressOf Client_KeepAlive

Private Sub Client_ServerConnected(sender As Object, e As EventArgs)
    Console.WriteLine(DateTime.Now.ToLocalTime() & " Session connected")
End Sub

Private Sub Client_ServerConnectionLost(sender As Object, e As EventArgs)
    Console.WriteLine(DateTime.Now.ToLocalTime() & " Session connection lost")
End Sub

Private Sub Client_KeepAlive(session As Session, e As KeepAliveEventArgs)
    'catch the keepalive event'
End Sub
```

Jetzt können je nach Wunsch über die Clientinstanz z.B. Daten gelesen, geschrieben, überwacht oder der Server gebrowst werden.
In der Default-Einstellung verbindet und trennt sich die Client-Instanz selbständig.

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Browsen über Nodes

Browsen mit Pfad-Angabe

Mit dem PLCcom.Opc.Ua.Sdk wurde auch das Browsen über bereitgestellte Servernodes vereinfacht. Zu diesem Zweck wird der Browse-Befehl bereitgestellt und mittels einer konfigurierten Clientinstanz ausgeführt. Die Befehle

- GetReferenceDescriptionByNodeId ,
- GetReferenceDescriptionByPath und
- GetNodeIdByPath

dienen zur Umwandlung eines „Browse Pfades“ zu einer NodeID oder zu einer ReferenceDescription bzw. zur Umwandlung einer NodeId zu einer ReferenceDescription.

Im Beispiel wird vorwärts über den Knoten "Objects.Server" gebrowst. Erstellen Sie zuerst ein BrowseDescription Objekt:

```
//C#
//Set start NodeId by path
NodeId sourceNode = client.GetNodeIdByPath("Objects.Data.Dynamic.Scalar");

// find all of the components of the node.
BrowseDescription nodeToBrowse1 = new BrowseDescription();

nodeToBrowse1.NodeId = sourceNode;
nodeToBrowse1.BrowseDirection = BrowseDirection.Forward;
nodeToBrowse1.ReferenceTypeId = ReferenceTypeIds.Aggregates;
nodeToBrowse1.IncludeSubtypes = true;
nodeToBrowse1.NodeClassMask = (uint)(NodeClass.Object | NodeClass.Variable);
nodeToBrowse1.ResultMask = (uint)BrowseResultMask.All;

// find all nodes organized by the node.
BrowseDescription nodeToBrowse2 = new BrowseDescription();

nodeToBrowse2.NodeId = sourceNode;
nodeToBrowse2.BrowseDirection = BrowseDirection.Forward;
nodeToBrowse2.ReferenceTypeId = ReferenceTypeIds.Organizes;
nodeToBrowse2.IncludeSubtypes = true;
nodeToBrowse2.NodeClassMask = (uint)(NodeClass.Object | NodeClass.Variable);
nodeToBrowse2.ResultMask = (uint)BrowseResultMask.All;

BrowseDescriptionCollection nodesToBrowse = new BrowseDescriptionCollection();
nodesToBrowse.Add(nodeToBrowse1);
nodesToBrowse.Add(nodeToBrowse2);
```

```
'Visual basic'
'Set start NodeId by path'
Dim sourceNode As NodeId = client.GetNodeIdByPath("Objects.Data.Dynamic.Scalar")

'Set start NodeId by path'
'find all of the components of the node. '
Dim nodeToBrowse1 As BrowseDescription = New BrowseDescription()
nodeToBrowse1.NodeId = sourceNode
nodeToBrowse1.BrowseDirection = BrowseDirection.Forward
nodeToBrowse1.ReferenceTypeId = ReferenceTypeIds.Aggregates
nodeToBrowse1.IncludeSubtypes = True
nodeToBrowse1.NodeClassMask = CUInt(NodeClass.Object Or NodeClass.Variable)
nodeToBrowse1.ResultMask = CUInt(BrowseResultMask.All)

' find all nodes organized by the node. '
Dim nodeToBrowse2 As BrowseDescription = New BrowseDescription()
nodeToBrowse2.NodeId = sourceNode
nodeToBrowse2.BrowseDirection = BrowseDirection.Forward
nodeToBrowse2.ReferenceTypeId = ReferenceTypeIds.Organizes
```

```
nodeToBrowse2.IncludeSubtypes = True
nodeToBrowse2.NodeClassMask = CUInt(NodeClass.Object Or NodeClass.Variable)
nodeToBrowse2.ResultMask = CUInt(BrowseResultMask.All)
Dim nodesToBrowse As BrowseDescriptionCollection = New BrowseDescriptionCollection()
nodesToBrowse.Add(nodeToBrowse1)
nodesToBrowse.Add(nodeToBrowse2)
```

Nun können Sie den Knoten `"Objects.Server"` browsen und erhalten ein ReferenceDescriptionCollection- Objekt mit dem Ergebnis der Operation zurück:

```
//C#
//now, browse the node
ReferenceDescriptionCollection rdc = client.BrowseFull(nodesToBrowse);

if (rdc.Count > 0)
{
    foreach (ReferenceDescription rd in rdc)
    {
        Console.WriteLine("Child NodeID found => "+ rd.NodeId+
            " NodeClass => " + rd.NodeClass.ToString()  +
            " BrowseName => " + rd.BrowseName.ToString() +
            " DisplayName => " + rd.DisplayName.ToString());
    }
}
```

```
'Visual Basic'
'now, browse the node'
Dim rdc As ReferenceDescriptionCollection = client.BrowseFull(nodesToBrowse)

If rdc.Count > 0 Then

    For Each rd As ReferenceDescription In rdc
        Console.WriteLine("Child NodeID found => " & rd.NodeId.ToString() &
                  " NodeClass => " & rd.NodeClass.ToString() &
                  " BrowseName => " & rd.BrowseName.ToString() &
                  " DisplayName => " & rd.DisplayName.ToString())
    Next
End If
```

Browsen mit NodeId-Angabe

Im Beispiel wird vorwärts über den Knoten Objects gebrowst. Hierzu übergeben wir die NodeId `ObjectIds.ObjectsFolder` an die BrowseDescription:

```
//C#
//Set start
NodeId sourceNode = ObjectIds.ObjectsFolder

// find all of the components of the node.
BrowseDescription nodeToBrowse1 = new BrowseDescription();

nodeToBrowse1.NodeId = sourceNode;
nodeToBrowse1.BrowseDirection = BrowseDirection.Forward;
nodeToBrowse1.ReferenceTypeId = ReferenceTypeIds.Aggregates;
nodeToBrowse1.IncludeSubtypes = true;
nodeToBrowse1.NodeClassMask = (uint)(NodeClass.Object | NodeClass.Variable);
nodeToBrowse1.ResultMask = (uint)BrowseResultMask.All;

// find all nodes organized by the node.
BrowseDescription nodeToBrowse2 = new BrowseDescription();

nodeToBrowse2.NodeId = sourceNode;
nodeToBrowse2.BrowseDirection = BrowseDirection.Forward;
nodeToBrowse2.ReferenceTypeId = ReferenceTypeIds.Organizes;
nodeToBrowse2.IncludeSubtypes = true;
nodeToBrowse2.NodeClassMask = (uint)(NodeClass.Object | NodeClass.Variable);
nodeToBrowse2.ResultMask = (uint)BrowseResultMask.All;

BrowseDescriptionCollection nodesToBrowse = new BrowseDescriptionCollection();
nodesToBrowse.Add(nodeToBrowse1);
nodesToBrowse.Add(nodeToBrowse2);
```

```
'Visual basic'
'Set start NodeId'
Dim sourceNode As NodeId = ObjectIds.ObjectsFolder

'Set start NodeId by path'
'find all of the components of the node. '
Dim nodeToBrowse1 As BrowseDescription = New BrowseDescription()
nodeToBrowse1.NodeId = sourceNode
nodeToBrowse1.BrowseDirection = BrowseDirection.Forward
nodeToBrowse1.ReferenceTypeId = ReferenceTypeIds.Aggregates
nodeToBrowse1.IncludeSubtypes = True
nodeToBrowse1.NodeClassMask = CUInt(NodeClass.Object Or NodeClass.Variable)
nodeToBrowse1.ResultMask = CUInt(BrowseResultMask.All)

' find all nodes organized by the node. '
Dim nodeToBrowse2 As BrowseDescription = New BrowseDescription()
nodeToBrowse2.NodeId = sourceNode
nodeToBrowse2.BrowseDirection = BrowseDirection.Forward
nodeToBrowse2.ReferenceTypeId = ReferenceTypeIds.Organizes
nodeToBrowse2.IncludeSubtypes = True
nodeToBrowse2.NodeClassMask = CUInt(NodeClass.Object Or NodeClass.Variable)
nodeToBrowse2.ResultMask = CUInt(BrowseResultMask.All)
Dim nodesToBrowse As BrowseDescriptionCollection = New BrowseDescriptionCollection()
nodesToBrowse.Add(nodeToBrowse1)
nodesToBrowse.Add(nodeToBrowse2)
```

Nun können Sie den Knoten "Objects" browsen und erhalten ein ReferenceDescriptionCollection-Objekt mit dem Ergebnis der Operation zurück:

```
//C#
//now, browse the node
ReferenceDescriptionCollection rdc = client.BrowseFull(nodesToBrowse);

if (rdc.Count > 0)
{
    foreach (ReferenceDescription rd in rdc)
    {
        Console.WriteLine("Child NodeID found => "+ rd.NodeId+
            " NodeClass => " + rd.NodeClass.ToString()  +
            " BrowseName => " + rd.BrowseName.ToString() +
            " DisplayName => " + rd.DisplayName.ToString());
    }
}
```

```
'Visual Basic'
'now, browse the node'
Dim rdc As ReferenceDescriptionCollection = client.BrowseFull(nodesToBrowse)

If rdc.Count > 0 Then

    For Each rd As ReferenceDescription In rdc
        Console.WriteLine("Child NodeID found => " & rd.NodeId.ToString() &
                  " NodeClass => " & rd.NodeClass.ToString() &
                  " BrowseName => " & rd.BrowseName.ToString() &
                  " DisplayName => " & rd.DisplayName.ToString())
    Next
End If
```

Lesen und Schreiben von Werten

Das eigentliche Lesen und Schreiben von Werten kann jeweils mit einer einzelnen Zeile Code durchgeführt werden. Voraussetzung ist eine konfigurierte Clientinstanz (siehe oben).

Per Default verbindet und trennt sich das PLCcom.Opc.Ua.Sdk automatisch zum Server und überwacht auch deren Verbindung.

Wie Sie es bereits kennengelernt haben, können Sie auch beim Monitoring den Node einfach mit dem kompletten Browse-Namen adressieren, die Umrechnung zur NodeID wird im Toolkit automatisch im Hintergrund durchgeführt.

Die nachfolgenden Beispiele veranschaulicht das Lesen und Schreiben an einem Opc-Ua-Server.

Vorgehensweise synchrones Lesen von Daten aus Knoten

Das nachfolgende Beispiel demonstriert das synchrone Lesen von Knoten eines OPC-UA-Servers. Mit dem Parameter AttributeId können Sie angeben, welches Attribut Sie lesen möchten:

```
//C#
ReadValueId nodeToRead = new ReadValueId();
nodeToRead.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int32Value");
nodeToRead.AttributeId = Attributes.Value;

//reading the node synchronous
DataValue readresults = client.Read(nodeToRead);
```

```
'Visual Basic'
ReadValueId nodeToRead = New ReadValueId()
nodeToRead.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int32Value")
nodeToRead.AttributeId = Attributes.Value

'reading the nodes synchronous'
Dim readresults As DataValueCollection = client.Read(nodeToRead)
```

Vorgehensweise synchrones Schreiben von Daten in Knoten

Das nachfolgende Beispiel demonstriert das synchrone Schreiben auf Knoten eines OPC-UA-Servers. Mit dem Parameter AttributeId können Sie angeben, welches Attribut Sie schreiben möchten:

```
//C#
//write value 123 to OPC UA Node and receive the status code

WriteValue nodeToWrite = new WriteValue();
nodeToWrite.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int64Value");
nodeToWrite.AttributeId = Attributes.Value;
nodeToWrite.Value = new DataValue((Int64)(123));

//writing the node synchronous
StatusCode writeResult = client.Write(nodeToWrite);
```

```
'Visual Basic'
'write value 123 to OPC UA Node and receive the status code'
Dim nodeToWrite As New WriteValue()
nodeToWrite.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int64Value")
nodeToWrite.AttributeId = Attributes.Value
nodeToWrite.Value = New DataValue(CLng(123))

'writing the nodes synchronous'
Dim writeResult As StatusCode = client.Write(nodeToWrite)
```

Vorgehensweise asynchrones Lesen von Daten aus Knoten

Das nachfolgende Beispiel demonstriert das asynchrone Lesen von Knoten eines OPC-UA-Servers.

Mit dem BeginRead-Befehl leiten Sie den asynchronen Lesevorgang ein und übergeben auch die entsprechende Methode zur Datenauswertung. Mit dem Parameter AttributeId können Sie angeben, welches Attribut Sie lesen möchten:

```
//C#
private void ReadData()
{
   ReadValueId nodeToRead = new ReadValueId();
   nodeToRead.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int32Value");
   nodeToRead.AttributeId = Attributes.Value;

   //reading the nodes asynchronous
   client.BeginRead(nodeToRead, GetReadAsyncResult, nodesToRead);

}

private void GetReadAsyncResult(IAsyncResult res)
{
    DataValueCollection values = null;
    DiagnosticInfoCollection diagnostic = null;
    ResponseHeader readresults = client.EndRead(res, out values, out diagnostic);

    ReadValueIdCollection req = res.AsyncState as ReadValueIdCollection;

    if (req != null)
    {
        for (int i = 0; i < values.Count; i++)
        {
            Console.WriteLine("asynchronous read result " +
                req[i].NodeId.ToString() + " Value => " +
                values[i].ToString() + " StatusCode => " +
                values[i].StatusCode.ToString());
        }
    }
}
```

```
'Visual Basic'
Private Sub ReadData ()

   ReadValueId nodeToRead = New ReadValueId()
   nodeToRead.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int32Value")
   nodeToRead.AttributeId = Attributes.Value

         'reading the nodes asynchronous'
   client.BeginRead(nodesToRead, New AsyncCallback(AddressOf GetReadAsyncResult), nodesToRead)

End Sub

Private Sub GetReadAsyncResult(ByVal res As IAsyncResult)
    Dim values As DataValueCollection = Nothing
    Dim diagnostic As DiagnosticInfoCollection = Nothing
    Dim readresults As ResponseHeader = client.EndRead(res, values, diagnostic)
    Dim req As ReadValueIdCollection = TryCast(res.AsyncState, ReadValueIdCollection)

    If req IsNot Nothing Then

        For i As Integer = 0 To values.Count - 1
            Console.WriteLine("asynchronous read result " & req(i).NodeId.ToString() &
                              " Value => " & values(i).ToString() &
                              "  StatusCode => " & values(i).StatusCode.ToString())
        Next
    End If
End Sub
```

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Vorgehensweise asynchrones Schreiben von Daten in Knoten

Das nachfolgende Beispiel demonstriert das asynchrone Schreiben auf Knoten eines OPC-UA-Servers. Mit dem Parameter AttributeId können Sie angeben, welches Attribut Sie schreiben möchten:

Mit dem BeginWrite-Befehl leiten Sie den asynchronen Schreibvorgang ein und übergeben auch die entsprechende Methode zur Datenauswertung. Mit dem Parameter AttributeId können Sie angeben, welches Attribut Sie schreiben möchten:

```
//C#
private void WriteData()
{
   //write value 123 to OPC UA Node and receive the status code
   WriteValue nodeToWrite = new WriteValue();
   nodeToWrite.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int64Value");
   nodeToWrite.AttributeId = Attributes.Value;
   nodeToWrite.Value = new DataValue((Int64)(123));

   //writing the nodes asynchronous
   client.BeginWrite(nodeToWrite, GetWriteAsyncResult, nodesToWrite);
}

private void GetWriteAsyncResult(IAsyncResult res)
{
    StatusCodeCollection statuscodes = null;
    DiagnosticInfoCollection diagnostic = null;
    ResponseHeader readresults = client.EndWrite(res, out statuscodes, out diagnostic);

    WriteValueCollection req = res.AsyncState as WriteValueCollection;
    if (req != null)
    {
        for (int i = 0; i < statuscodes.Count; i++)
        {
            Console.WriteLine("asynchronous write result " + req[i].NodeId.ToString() +
                " Value => " + req[i].Value.ToString() +
                " StatusCode => " + statuscodes[i].ToString());
        }
    }
}
```

```
'Visual Basic'
Private Sub WriteData()

 'write value 123 to OPC UA Node and receive the status code'
 Dim nodeToWrite As New WriteValue()
 nodeToWrite.NodeId = client.GetNodeIdByPath("Objects.Data.Static.Scalar.Int64Value")
 nodeToWrite.AttributeId = Attributes.Value
 nodeToWrite.Value = New DataValue(CLng(123))

  'writing the nodes asynchronous'
 client.BeginWrite(nodeToWrite, New AsyncCallback(AddressOf GetWriteAsyncResult), nodeToWrite)
End Sub

Private Sub GetWriteAsyncResult(ByVal res As IAsyncResult)
    Dim statuscodes As StatusCodeCollection = Nothing
    Dim diagnostic As DiagnosticInfoCollection = Nothing
    Dim readresults As ResponseHeader = client.EndWrite(res, statuscodes, diagnostic)
    Dim req As WriteValueCollection = TryCast(res.AsyncState, WriteValueCollection)

    If req IsNot Nothing Then
        For i As Integer = 0 To statuscodes.Count - 1
            Console.WriteLine("asynchronous write result " & req(i).NodeId.ToString() &
                              " Value => " & req(i).Value.ToString() &
                              " StatusCode => " & statuscodes(i).ToString())
        Next
    End If
End Sub
```

Verfügbare Attribute zum Lesen oder Schreiben

Innerhalb der PLCcom.Opc.Ua.Sdk stehen die möglichen Attribute zum Lesen oder Schreiben vorkonfiguriert in der Klasse PLCcom.Opc.Ua.Attributes als Konstanten zur Verfügung.

Die Klasse beinhaltet folgende Objekte:

```
namespace PLCcom.Opc.Ua
{
    public static partial class Attributes
    {
        // The canonical identifier for the node.
        public const uint NodeId = 1;

        // The class of the node.
        public const uint NodeClass = 2;

        // A non-localized, human readable name for the node.
        public const uint BrowseName = 3;

        // A localized, human readable name for the node.
        public const uint DisplayName = 4;

        // A localized description for the node.
        public const uint Description = 5;

        // Indicates which attributes are writeable.
        public const uint WriteMask = 6;

        // Indicates which attributes are writeable by the current user.
        public const uint UserWriteMask = 7;

        // Indicates that a type node may not be instantiated.
        public const uint IsAbstract = 8;

        // Indicates that forward and inverse references have the same meaning.
        public const uint Symmetric = 9;

        // The browse name for an inverse reference.
        public const uint InverseName = 10;

        // Indicates that following forward references within a view will not cause a loop.
        public const uint ContainsNoLoops = 11;

        // Indicates that the node can be used to subscribe to events.
        public const uint EventNotifier = 12;

        // The value of a variable.
        public const uint Value = 13;

        // The node id of the data type for the variable value.
        public const uint DataType = 14;

        // The number of dimensions in the value.
        public const uint ValueRank = 15;

        // The length for each dimension of an array value.
        public const uint ArrayDimensions = 16;

        // How a variable may be accessed.
        public const uint AccessLevel = 17;

        // How a variable may be accessed after taking the user's access rights into account.
        public const uint UserAccessLevel = 18;

        // Specifies (in milliseconds) how fast the server can reasonably sample the value for
 changes.
        public const uint MinimumSamplingInterval = 19;

       // Specifies whether the server is actively collecting historical data for the variable
```

```
        public const uint Historizing = 20;

        // Whether the method can be called.
        public const uint Executable = 21;

        // Whether the method can be called by the current user.
        public const uint UserExecutable = 22;

        // Provides the metadata and encoding information for custom DataTypes.
        public const uint DataTypeDefinition = 23;

        // The permissions for the node granted to roles.
        public const uint RolePermissions = 24;

        /// The subset of permissions available for the roles available to the current session
        public const uint UserRolePermissions = 25;

        // The access restrictions assigned to the node.
        public const uint AccessRestrictions = 26;

        // How a variable may be accessed.
        public const uint AccessLevelEx = 27;
    }
}
```

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Monitoring von Knoten

Auch die Monitoring-Funktionalitäten des PLCcom.Opc.Ua.Sdk wurden für den Entwickler äußerst komfortabel und einfach zur Verfügung gestellt.
Voraussetzung ist eine konfigurierte Clientinstanz (siehe oben).
Per Default verbindet und trennt sich das PLCcom.Opc.Ua.Sdk automatisch zum Server und überwacht auch deren Verbindung. Über anstehende Verbindungsabbrüche werden Sie per Event informiert.

Wie Sie es bereits gewohnt sind, können Sie auch beim Monitoring den Node mit dem kompletten Browse-Namen adressieren, die Umrechnung zur NodeID wird in der PLCcom.Opc.Ua.Sdk automatisch im Hintergrund durchgeführt.

Erzeugen von Subscription-Instanzen

Um eine neue Subscription erzeugen zu können, muss der entsprechende Konstuktior aufgerufen werden. Nachfolgend können Sie die gewünschten Eigenschaften setzen und vorhandene Events zur Überwachung der Subscription registrieren. Abschließend muss die Subscription der UaClient-Instanz übergeben werden.

```
//C#
//create a new subscription
Subscription subscription = new Subscription();

subscription.PublishingInterval = 1000;
subscription.DisplayName = "mySubsription";

//register subscription events
subscription.StateChanged += Subscription_StateChanged;
subscription.PublishStatusChanged += Subscription_PublishStatusChanged;

//add new subscription to client
client.AddSubscription(subscription);
```

```
'Visual Basic'
'create a new subscription'
Dim subscription As Subscription = New Subscription()
subscription.PublishingInterval = 1000
subscription.PublishingEnabled = False
subscription.DisplayName = "mySubsription"

'register subscription events'
AddHandler subscription.StateChanged, AddressOf Subscription_StateChanged
AddHandler subscription.PublishStatusChanged, AddressOf Subscription_PublishStatusChanged

'add new subscription to client'
client.AddSubscription(subscription)
```

Ändern von Parametern einer Subscription

Über die Methoden des Sdks lassen sich auch Subscriptions ändern.
Im nachfolgend gezeigten Beispiel wird der ‚publishing interval' auf 500ms und geändert und der ‚publishing mode' aktiviert.

```
//C#
//enable publishing mode of subscription and set PublishingInterval
subscription.PublishingInterval = 500;
subscription.SetPublishingMode(true);
subscription.Modify();
```

```
'Visual Basic'
'enable publishing mode of subscription And set PublishingInterval'
subscription.PublishingInterval = 500
subscription.SetPublishingMode(True)
subscription.Modify()
```

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Management von MonitoredItems

Ein MonitoredItems-Object entspricht einem zu überwachenden Knoten eines Opc Ua Servers. Ein oder viele MonitoredItems werden in einer Subscription logisch gebündelt. Über das Subscription-Objekt können MonitoredItems-Objekte erzeugt, verändert oder gelöscht werden.

Erzeugen Sie ein Subscription-Objekt. Danach erzeugen Sie ein oder mehrere MonitoredItems-Objekte und füge diese Objekte Ihrer Subscription hinzu.
Das „Notification-Event“ verknüpfen Sie mit einer bereitzustellenden „Notification-Methode“ (siehe Beispiel). Über das „Notification-Event“ werden Ihnen nun die Werteänderungen des MonitoredItem-Objektes zur Verfügung gestellt.

Zum Abschluss rufen Sie die Methode subscription.ApplyChanges().

```
//C#
//create a new subscription
Subscription subscription = new Subscription();

subscription.PublishingInterval = 1000;
subscription.DisplayName = "mySubsription";

//register subscription events
subscription.StateChanged += Subscription_StateChanged;
subscription.PublishStatusChanged += Subscription_PublishStatusChanged;

//add new subscription to client
client.AddSubscription(subscription);

//Create a monitoring item and add to the subscription
NodeId nodeId = client.GetNodeIdByPath("Objects.Data.Dynamic.Scalar.Int64Value");
MonitoredItem monitoredItem = new MonitoredItem(subscription.DefaultItem)
                              {
                                  StartNodeId = nodeId,
                                  SamplingInterval = 500,
                                  QueueSize = UInt32.MaxValue,
                                  DisplayName = nodeId.ToString()
                              };

//register monitoring event
monitoredItem.Notification += Client_MonitorNotification;
//add Item to subscription
subscription.AddItem(monitoredItem);

//apply changes
subscription.ApplyChanges();

...

private void Client_MonitorNotification(MonitoredItem monitoredItem, MonitoredItemNotification
EventArgs e)
{
    MonitoredItemNotification notification = e.NotificationValue as MonitoredItemNotification;
    Console.WriteLine(monitoredItem.StartNodeId.Identifier +
        " Value: " + notification.Value +
        " Status: " + notification.Value.StatusCode.ToString());
}
```

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

```
'Visual Basic'
'create a new subscription'
Dim subscription As Subscription = New Subscription()
subscription.PublishingInterval = 1000
subscription.PublishingEnabled = False
subscription.DisplayName = "mySubsription"

'register subscription events'
AddHandler subscription.StateChanged, AddressOf Subscription_StateChanged
AddHandler subscription.PublishStatusChanged, AddressOf Subscription_PublishStatusChanged

'add new subscription to client'
client.AddSubscription(subscription)

'Create a monitoring item and add to the subscription'
Dim nodeId As NodeId = client.GetNodeIdByPath("Objects.Data.Dynamic.Scalar.Int64Value")
Dim monitoredItem As MonitoredItem = New MonitoredItem(subscription.DefaultItem) With {
                                    .StartNodeId = nodeId,
                                    .SamplingInterval = 500,
                                    .QueueSize = UInteger.MaxValue,
                                    .DisplayName = nodeId.ToString()
                                }

'register monitoring event'
AddHandler monitoredItem.Notification, AddressOf Client_MonitorNotification

'add Item to subscription'
subscription.AddItem(monitoredItem)

'apply changes'
subscription.ApplyChanges()

...

Private Sub Client_MonitorNotification(ByVal monitoredItem As MonitoredItem, ByVal e As Monito
redItemNotificationEventArgs)
    Dim notification As MonitoredItemNotification = TryCast(e.NotificationValue, MonitoredItem
Notification)
    Console.WriteLine(monitoredItem.StartNodeId.Identifier &
                      " Value: " & notification.Value &
                      " Status: " & notification.Value.StatusCode.ToString())
End Sub
```

Call Aufrufe

Mit einem Call-Aufruf, lassen sich vom Client aus im Server hinterlegte Methoden aufrufen bzw. ausführen.

Bitte beachten Sie:
Jeder Opc Ua Server unterstützt nur die speziell für ihn definierten Call-Aufrufe.
Welche spezifischen Call-Aufrufe ihr Opc-Ua-Server unterstützt, entnehmen Sie bitte der Beschreibung des Opc-Server-Herstellers.

Im Lieferumfang finden Sie innerhalb der Tutorials weitere Codebeispiele für die Ausführung von Call-Methoden mit der Übertragung von einfachen flachen Daten sowie komplexen Strukturen.

Nachfolgend die Definition der Datenstruktur zur Übergabe mit einem Methodenaufruf:

```
//C#
 /*
let´s starting a method call, step by step
In this simple case, we pass a simple structure named as 'DataStructure_One" constructed as
follows:

structure DataStructure_One =
{
  int myIntValue1,
  string myStringValue2,
  int myIntValue3,
  int myIntValue4,
  string myStringValue5
}

Object to which the method should be applied is named as "myObjectNode"
Method is named as "myMethodNode"
*/

int myIntValue1 = 1;
string myStringValue2 = "testvalue";
int myIntValue3 = 3333;
int myIntValue4 = 4444;
string myStringValue5 = "a_string_value";
```

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Nächster Schritt, Vorbereitung und Ausführung der Call-Methode:

```
//C#
//Call Methods

//create a Encoder instance
BinaryEncoder encoder = new BinaryEncoder(client.getMessageContext());

//put objects to encoder with given order
encoder.WriteInt32("", myIntValue1);
encoder.WriteString("", myStringValue2);
encoder.WriteInt32("", myIntValue3);
encoder.WriteInt32("", myIntValue4);

//read byte array from encoder
byte[] argumentByteArray = new byte[encoder.BaseStream.Length];
encoder.BaseStream.Position = 0;
encoder.BaseStream.Read(argumentByteArray, 0, argumentByteArray.Length);

//create an extension object and pass arguments to ExtensionObject.Body
ExtensionObject extensionObjectWithInputArguments = new ExtensionObject();
extensionObjectWithInputArguments.Body = argumentByteArray;

//set type of structure, create a new ExpandedNodeId by name and namespace
extensionObjectWithInputArguments.TypeId = new ExpandedNodeId("DataStructure_One", 3);

//create your InputArguments with extensionObject
VariantCollection inputArguments = new VariantCollection();
inputArguments.Add(new Variant(extensionObjectWithInputArguments));

//create a new NodeId for the Object to which the method should be applied by name and
//namespace
NodeId objectNode = new NodeId("myObjectNode", 3);

//create a new NodeId for the Method by name and namespace
NodeId methodNode = new NodeId("myMethodNode", 3);

//create a CallMethodRequest instance and pass your arguments
CallMethodRequest request = new CallMethodRequest();
request.ObjectId = objectNode;
request.MethodId = methodNode;
request.InputArguments = inputArguments;

//call your method
CallMethodResult result = client.Call(request);

//finaly evaluate your results,
if (StatusCode.IsGood(result.StatusCode))
{
    foreach (Variant outputArgument in result.OutputArguments)
    {
            if (outputArgument != Variant.Null)
                Console.WriteLine("output argument: " + outputArgument.ToString());
    }
}
```

Nachfolgend das gleiche Beispiel in Visual Basic:

```
'Visual Basic'
 'Structure DataStructure_One = '
'{'
'    int myIntValue1, '
'    string myStringValue2, '
'    int myIntValue3, '
'    int myIntValue4, '
'    string myStringValue5'
'}'

'Object to which the method should be applied Is named as "myObjectNode"'
'Method Is named as "myMethodNode"'


Dim myIntValue1 As Integer = 1
Dim myStringValue2 As String = "testvalue"
Dim myIntValue3 As Integer = 3333
Dim myIntValue4 As Integer = 4444
Dim myStringValue5 As String = "a_string_value"
```

Nächster Schritt, Vorbereitung und Ausführung der Call-Methode:

```
'Visual Basic'
'Call Methods'
'create a Encoder instance'
Dim encoder As BinaryEncoder = New BinaryEncoder(client.getMessageContext)

'put objects to encoder with given order'
encoder.WriteInt32("", myIntValue1)
encoder.WriteString("", myStringValue2)
encoder.WriteInt32("", myIntValue3)
encoder.WriteInt32("", myIntValue4)
encoder.WriteString("", myStringValue5)

'read byte array from encoder'
Dim argumentByteArray() As Byte = New Byte((encoder.BaseStream.Length) - 1) {}
encoder.BaseStream.Position = 0
encoder.BaseStream.Read(argumentByteArray, 0, argumentByteArray.Length)

'create an extension object and pass arguments to ExtensionObject.Body'
Dim extensionObjectWithInputArguments As ExtensionObject = New ExtensionObject()
extensionObjectWithInputArguments.Body = argumentByteArray

'set type of structure, create a new ExpandedNodeId by name and namespace'
extensionObjectWithInputArguments.TypeId = New ExpandedNodeId( _
New NodeId("DataStructure_One", 3))

'create your InputArguments with extensionObject'
Dim inputArguments As VariantCollection = New VariantCollection
inputArguments.Add(New PLCcom.Opc.Ua.Variant(extensionObjectWithInputArguments))

'create a new NodeId for the Object to which the method should be applied by name and '
'namespace'
Dim objectNode As NodeId = New NodeId("myObjectNode", 3)

'create a new NodeId for the Method by name and namespace'
Dim methodNode As NodeId = New NodeId("myMethodNode", 3)

'create a CallMethodRequest instance and pass your arguments'
Dim request As CallMethodRequest = New CallMethodRequest
request.ObjectId = objectNode
request.MethodId = methodNode
request.InputArguments = inputArguments

'call your method'
Dim result As CallMethodResult = client.Call(request)

'finaly evaluate your results'
If StatusCode.IsGood(result.StatusCode) Then

  For Each outputArgument As PLCcom.Opc.Ua.Variant In result.OutputArguments
      If (outputArgument <> PLCcom.Opc.Ua.Variant.Null) Then
          Console.WriteLine(("output argument: " + outputArgument.ToString))
      End If

  Next

Else
  Console.WriteLine(("Method call failed " + result.StatusCode.ToString))
End If
```

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Zertifikatsverwaltung

Für eine gesicherte Kommunikation zwischen OPC-UA-Server und OPC-UA-Client ist es möglich Zertifikate auszutauschen. Hierzu stellt das PLCcom.Opc.Ua.Sdk standardmäßig einen Zertifikatsstore zur Verfügung.
Der Pfad zum Zertifikatsstore wird in der Sessionkonfiguration gespeichert und kann über die Eigenschaft „CertificateStorePath“ eingesehen oder verändert werden. Per Default wird der Zertifikatsstore unter dem Pfad %ProgramData%\<Applikationsname>\CertificateStores abgelegt.

Die Validierung eines Zertifikates kann auch über das Event „CertificateValidation“ der UaClient-Instanz abgefangen und verarbeitet werden. Das Sdk akzeptiert in Default-Einstellung auch ungültige Server-Zertifikate. Sollte das nicht gewünscht sein, kann die Eigenschaft " AutoAcceptUntrustedCertificates“ innerhalb der Sessionkonfiguration auf den Wert „false“ gesetzt werden.

Zur besonderen Beachtung:
Mit dem Verbindungsversuch einer UaClient-Instanz wird automatisch der Zertifikatsstore auf ein passendes Zertifikat geprüft, sollte kein Zertifikat vorhanden sein, wird automatisch ein selbst signiertes Zertifikat erstellt. Dieses automatisch erstellte Zertifikat, enthält keine Informationen über eine vertrauenswürdige Zertifizierungsstelle. Sollte die Verwendung dieses Zertifikates trotzdem erwünscht sein, muss es serverseitig unter vertrauenswürdige Zertifizierungsstellen installiert werden.
Wenn eine gesicherte Verbindung mit Zertifikataustausch gewünscht ist, muss das Serverzertifikat im Zertifikatsstore des Clients und das Clientzertifikat im Server-Zertifikatsstore installiert sein. Dieser Zertifikatsaustausch muss manuell über das Filesystem durchgeführt werden.

```
//C#
//catching certificate validate event
client.CertificateValidation += clie_CertificateValidation;

...

void clie CertificateValidation(CertificateValidator sender,CertificateValidationEventArgs e)
{
    //external certificate validation
    if (ServiceResult.IsGood(e.Error))
        e.Accept = true;
    else if ((e.Error.StatusCode.Code == StatusCodes.BadCertificateUntrusted ||
        e.Error.StatusCode.Code == StatusCodes.BadCertificateChainIncomplete) &&
        autoAcceptUntrustedCertificates)
        e.Accept = true;
    else
    {
      throw new Exception(string.Format("Failed to validate certificate error code {0}: {1}",
                          e.Error.Code,
                          e.Error.AdditionalInfo));
    }
}
```

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

```
'Visual Basic'
 ' catching certificate validate event'  AddHandler client.CertificateValidation, AddressOf
client_CertificateValidation
…
Private Sub client_CertificateValidation(ByVal sender As CertificateValidator, ByVal e As Cert
ificateValidationEventArgs)
    'external certificate validation'
    If ServiceResult.IsGood(e.Error) Then
        e.Accept = True
    ElseIf (e.Error.StatusCode.Code = StatusCodes.BadCertificateUntrusted OrElse
        e.Error.StatusCode.Code = StatusCodes.BadCertificateChainIncomplete) AndAlso
        autoAcceptUntrustedCertificates Then
        e.Accept = True
    Else
        Throw New Exception("Failed to validate certificate with error code " &
                            e.Error.Code & " " & e.Error.AdditionalInfo)
    End If
End Sub
```

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Haben Sie Fragen oder Hinweise?

Rufen Sie uns bitte unter der Telefonnummer +49 421 98970330 an, oder senden Sie Ihre Frage oder Hinweise an support@indi-systems.de.

Wir werden Ihr Anliegen in kürzester Zeit bearbeiten oder direkt beantworten.

Indi.An Systems GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39